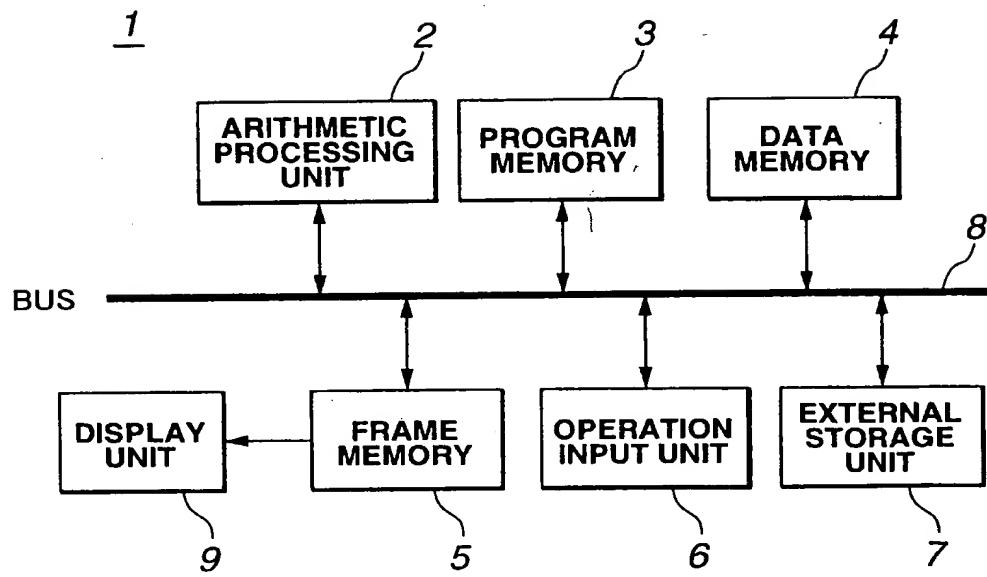
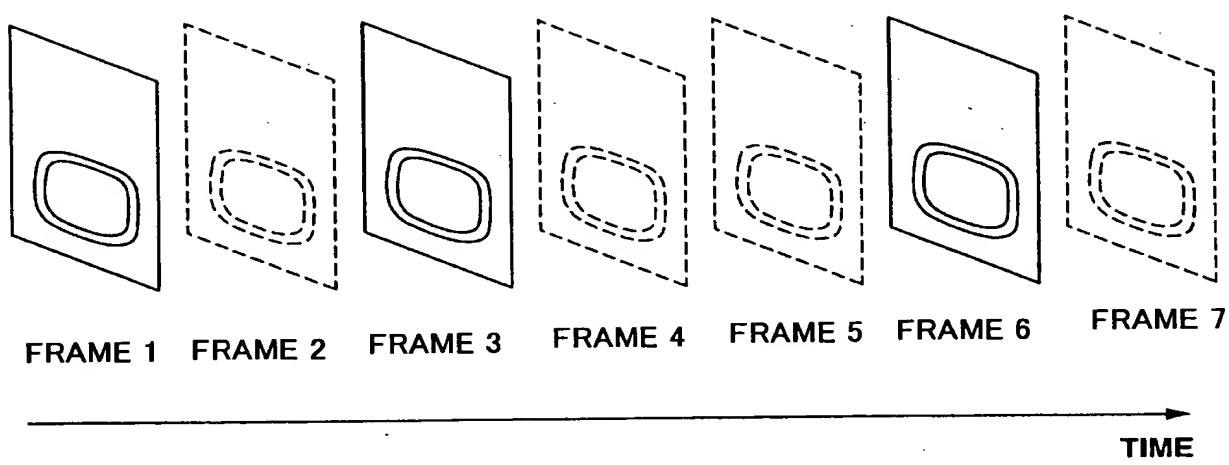


[FIG. 1]

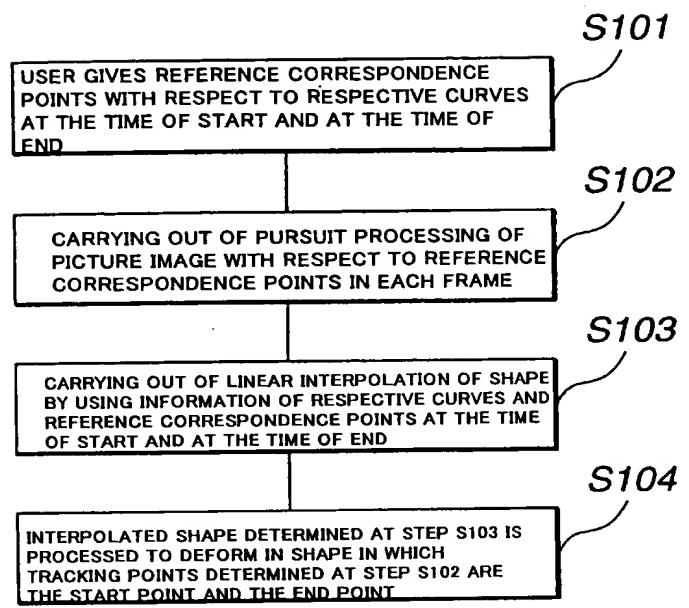


[FIG. 2]

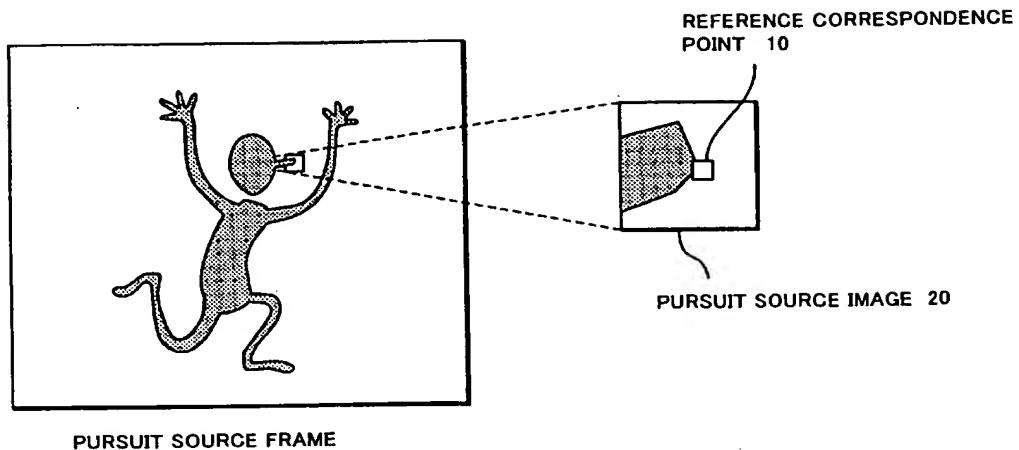
00000000000000000000000000000000



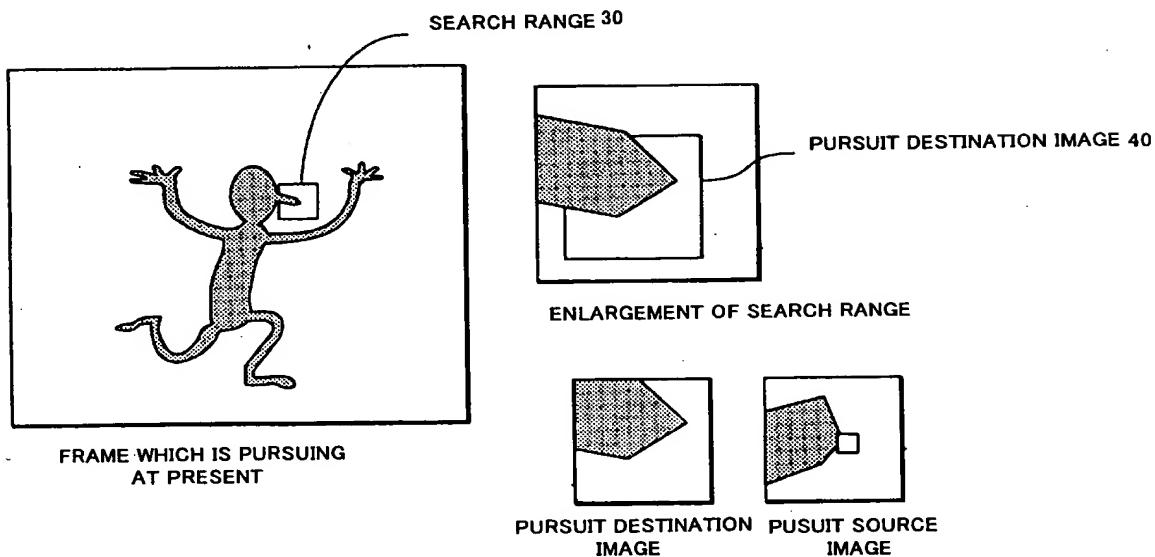
[FIG. 3]



[FIG. 4]



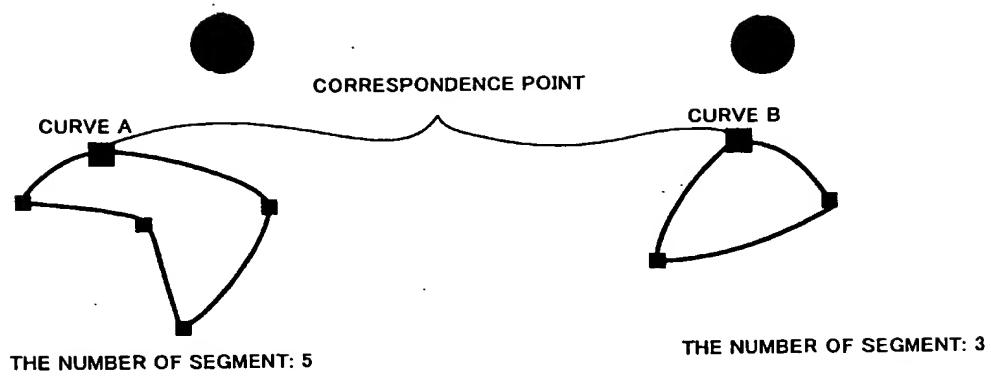
### PURSUIT SOURCE FRAME



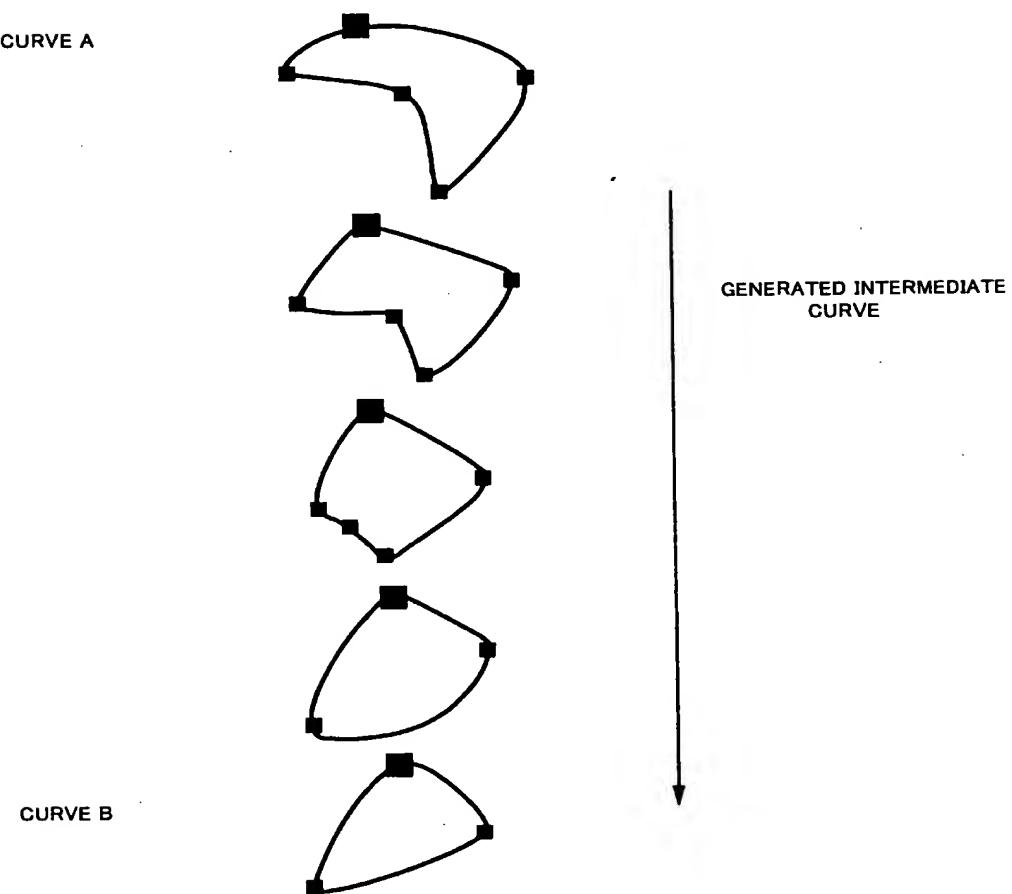
**FRAME WHICH IS PURSUING  
AT PRESENT**

PURSUIT DESTINATION PURSUIT SOURCE  
IMAGE IMAGE

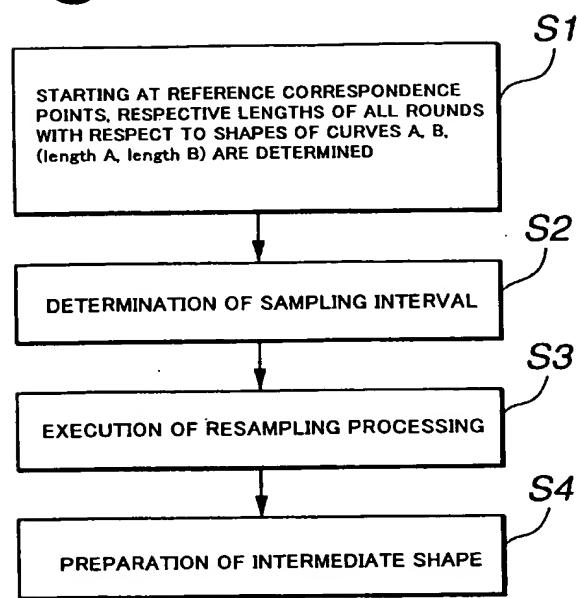
[FIG. 5]



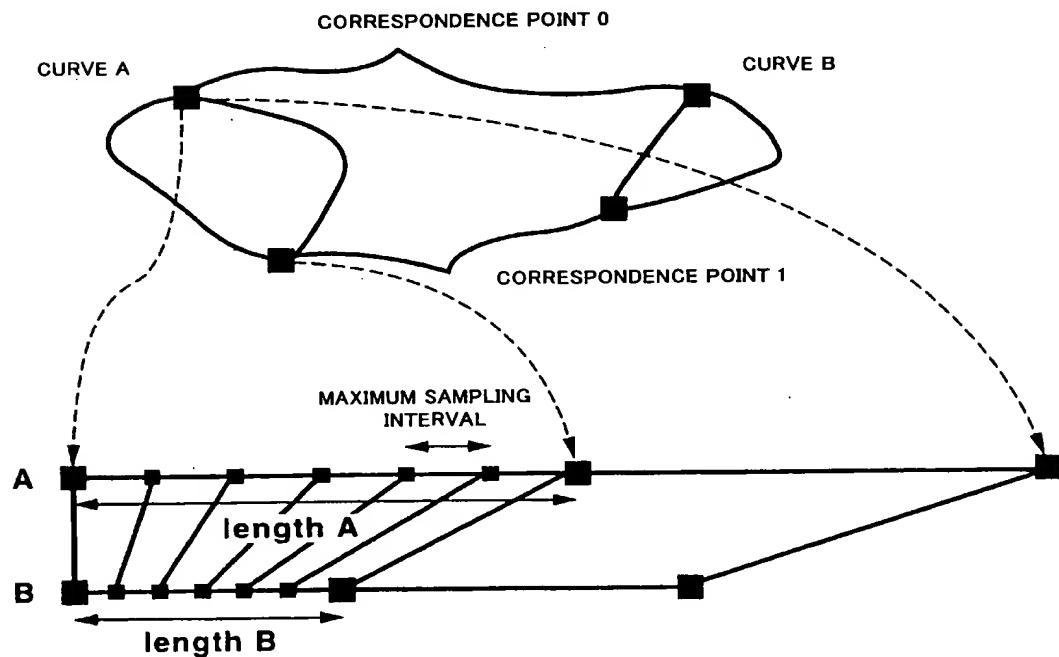
[FIG. 6]



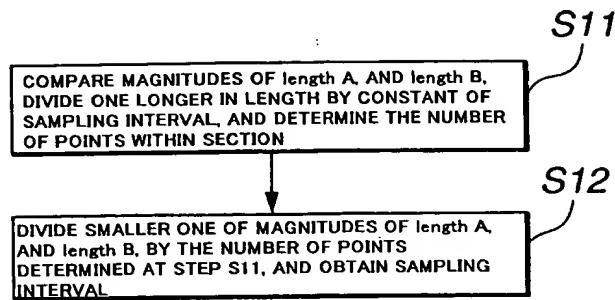
[FIG. 7]



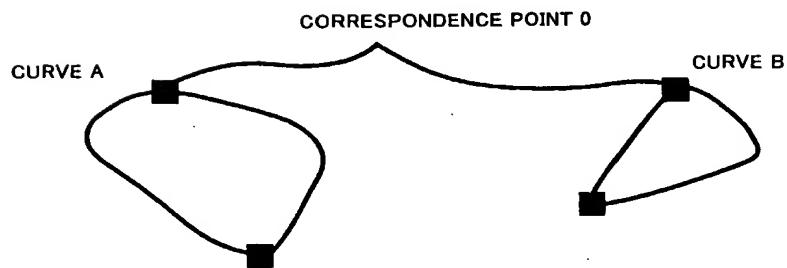
[FIG. 8]



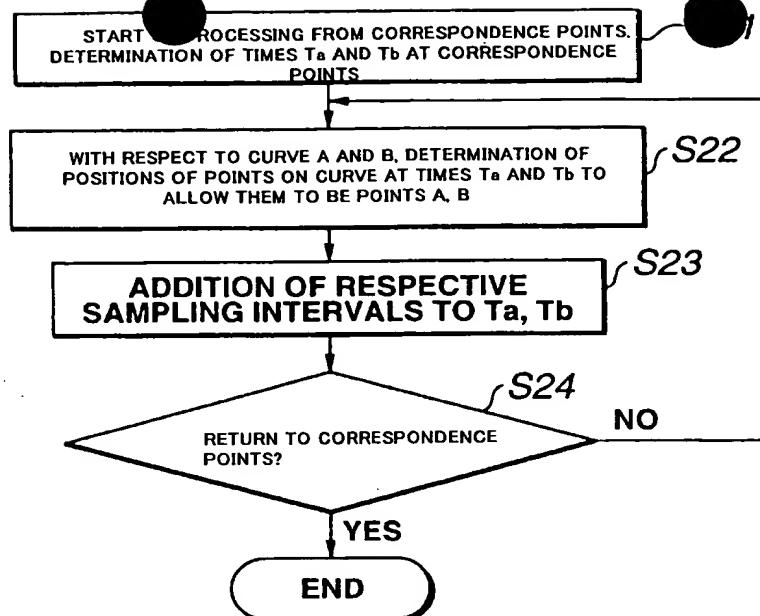
[FIG. 9]



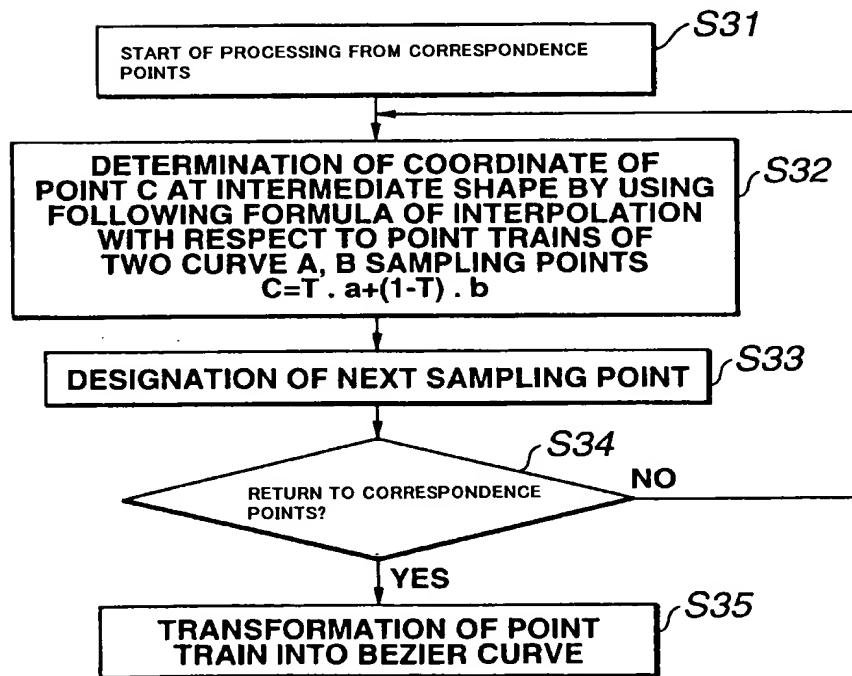
[FIG. 10]



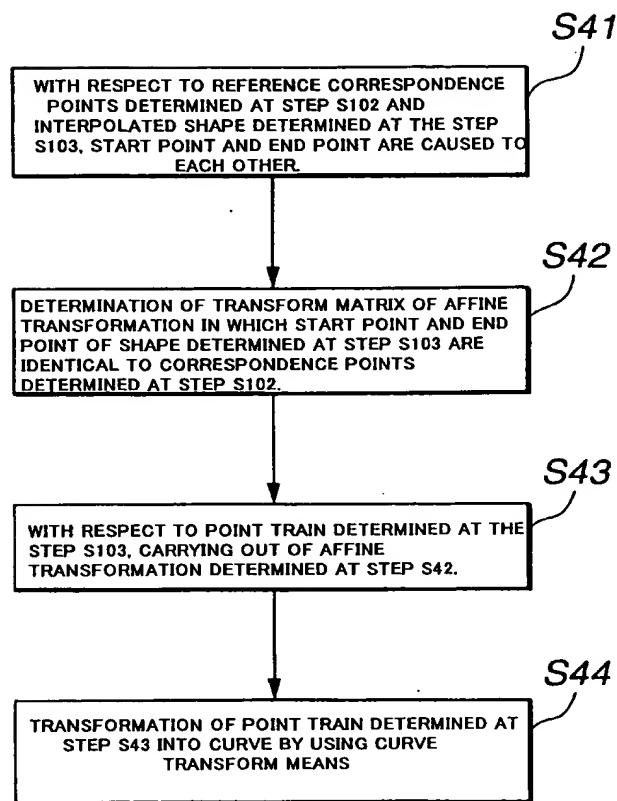
[FIG. 11]



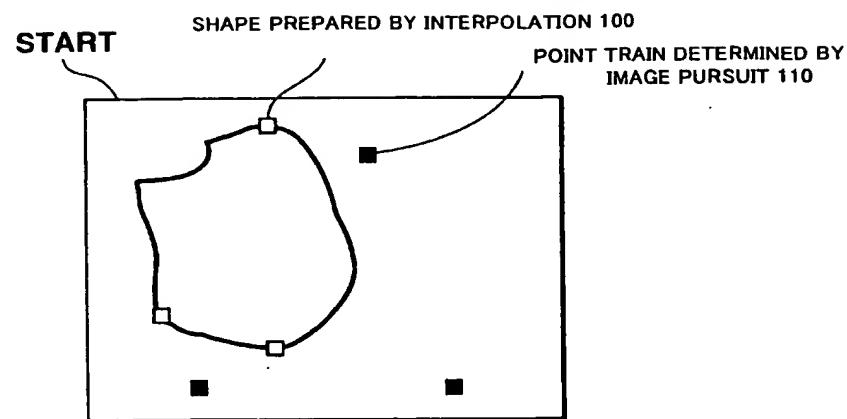
[FIG. 12]



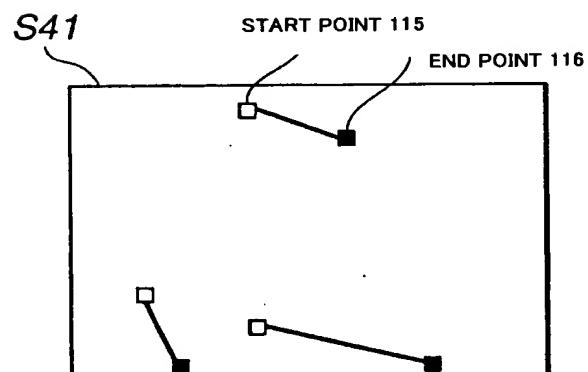
[FIG. 13]



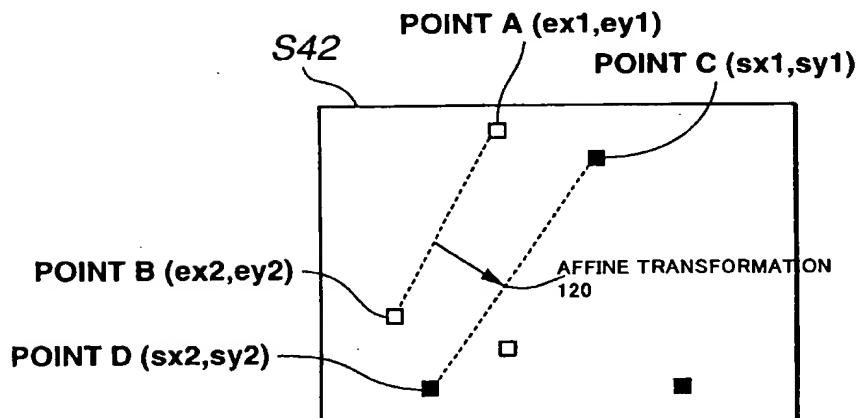
[FIG. 14]



SHAPE PREPARED BY INTERPOLATION AND  
POINT TRAIN DETERMINED BY IMAGE PURSUIT



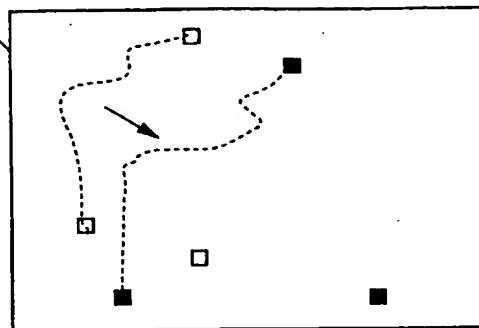
CORRESPONDENCE OF REFERENCE  
CORRESPONDENCE POINT



DETERMINE AFFINE TRANSFORMATION WHICH  
IS DEFORMED AS DESCRIBED ABOVE WITH  
RESPECT TO CORRESPONDENCE SECTION.

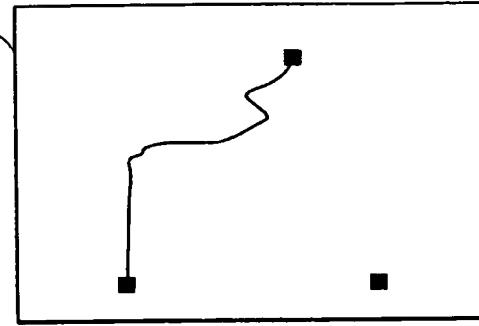
[FIG. 15]

S43



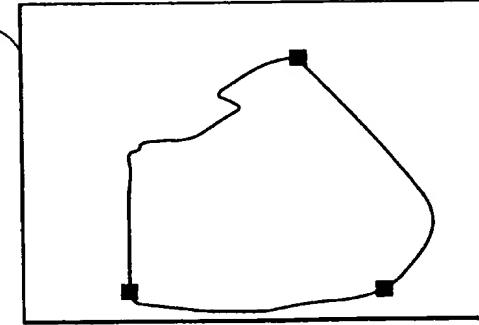
TRANSFORMING WITH RESPECT TO POINT TRAIN  
CONSTITUTING INTERMEDIATE SHAPE AS WELL

S44

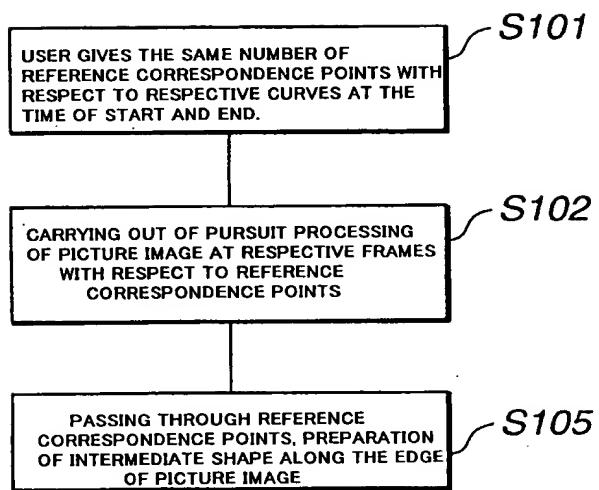


TRANSFORMING INTO BEZIER CURVE

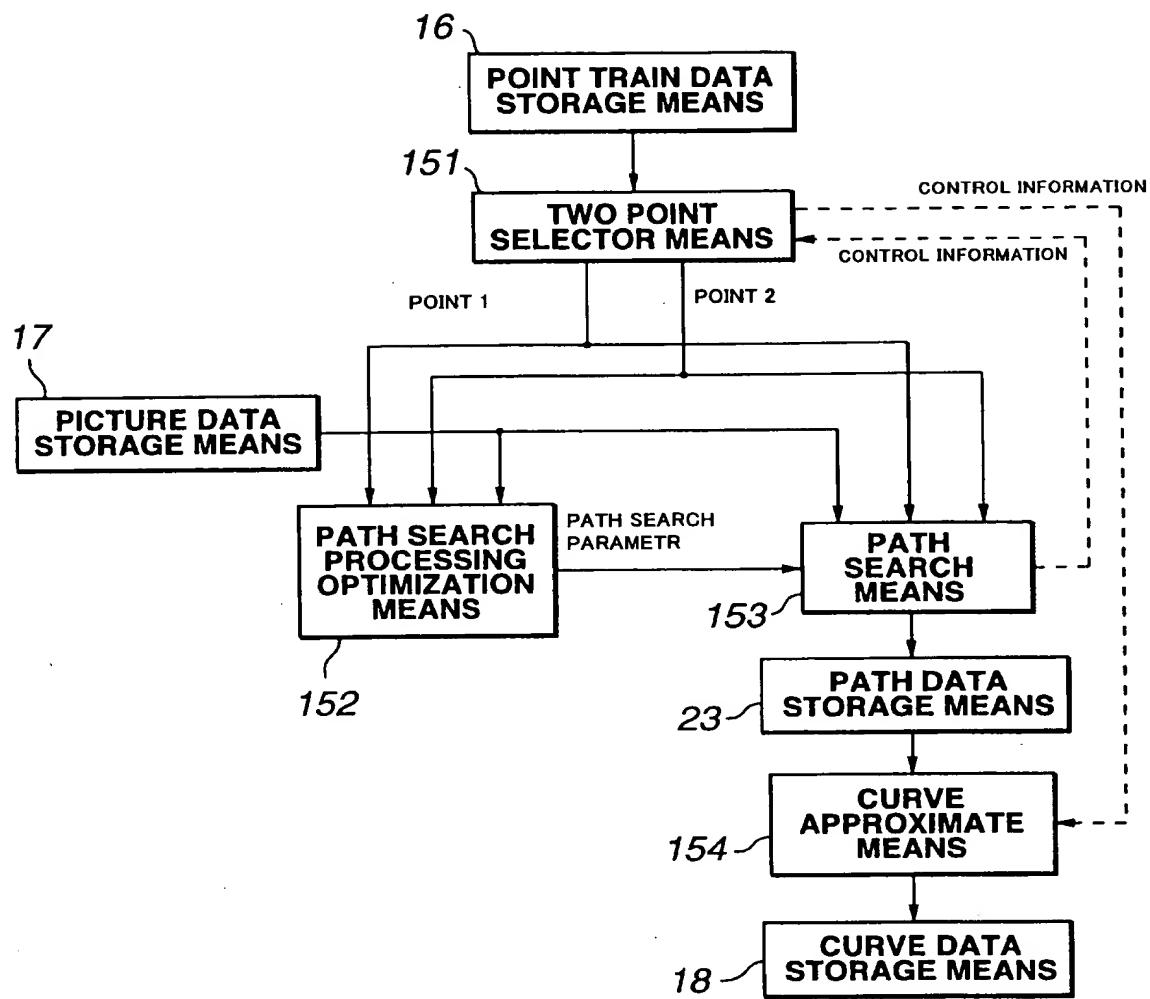
END



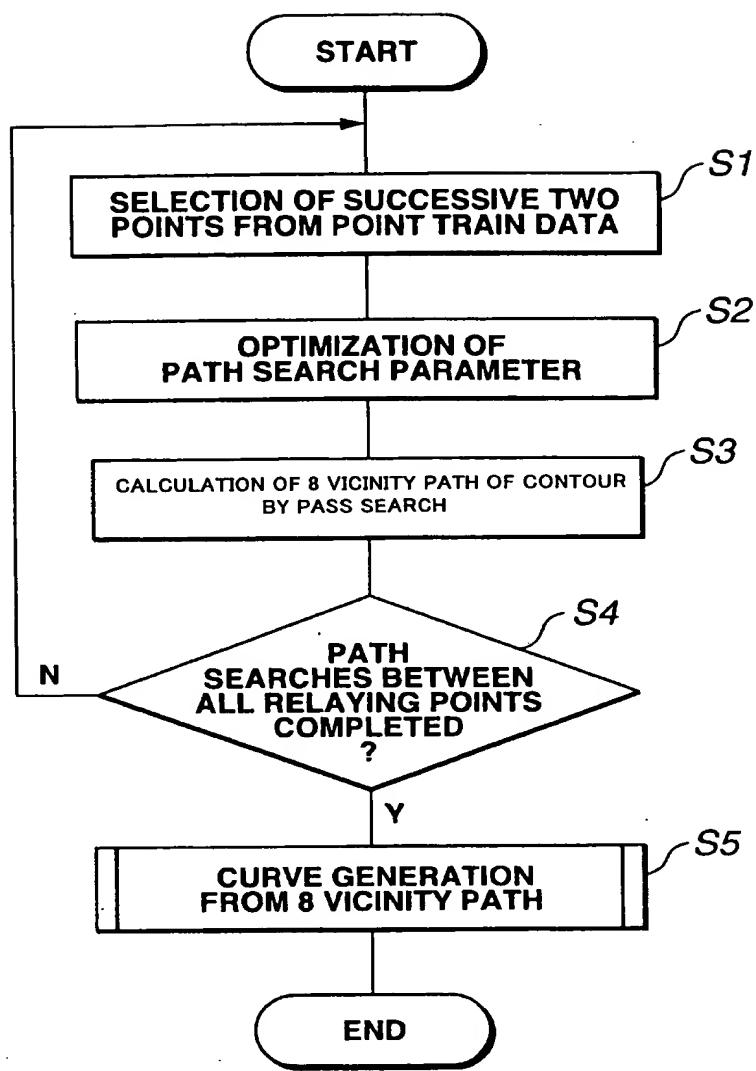
RESULT OF TRANSFORMING WITH RESPECT TO  
ALL SECTIONS



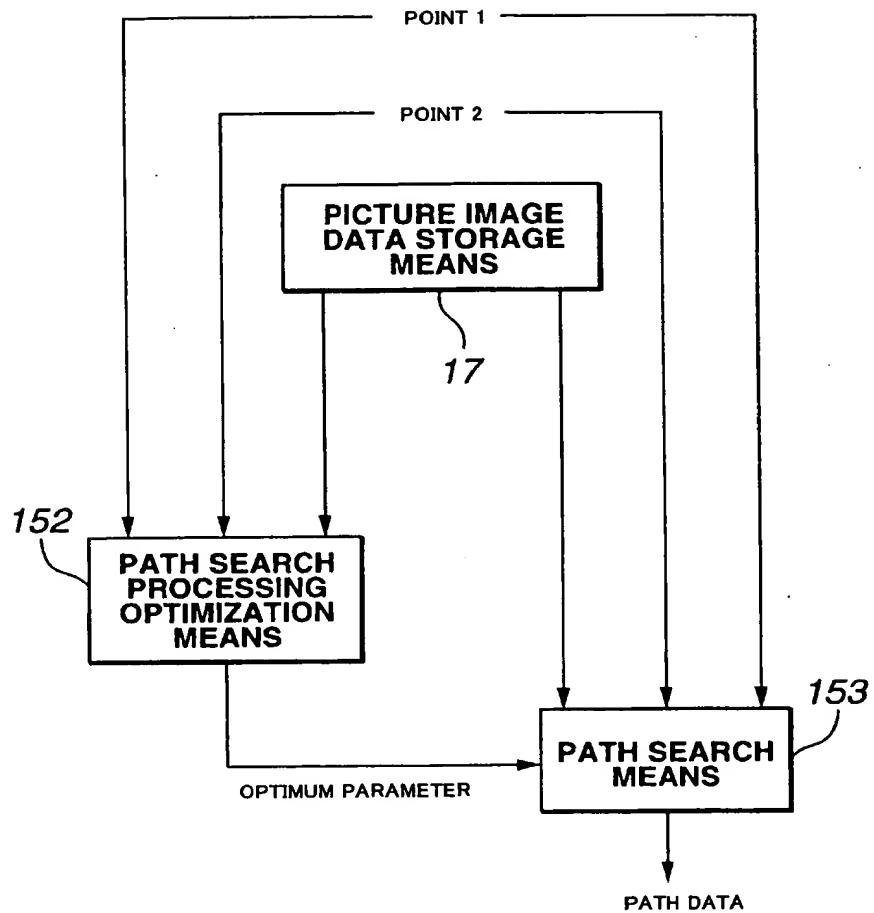
[FIG. 17]



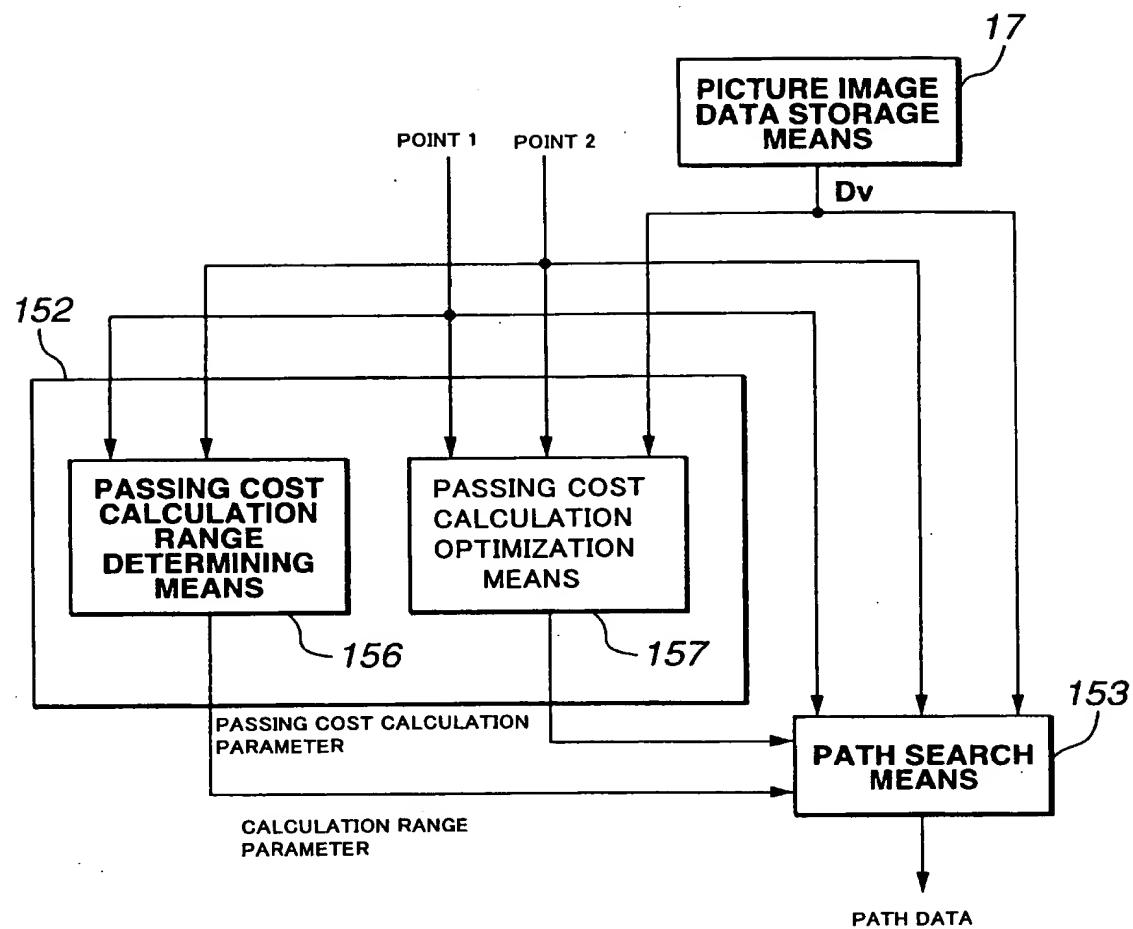
[FIG. 18]



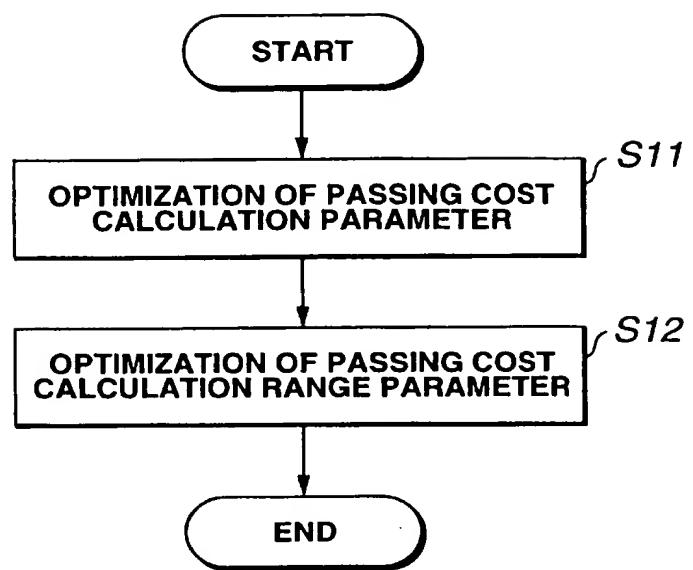
[FIG. 19]



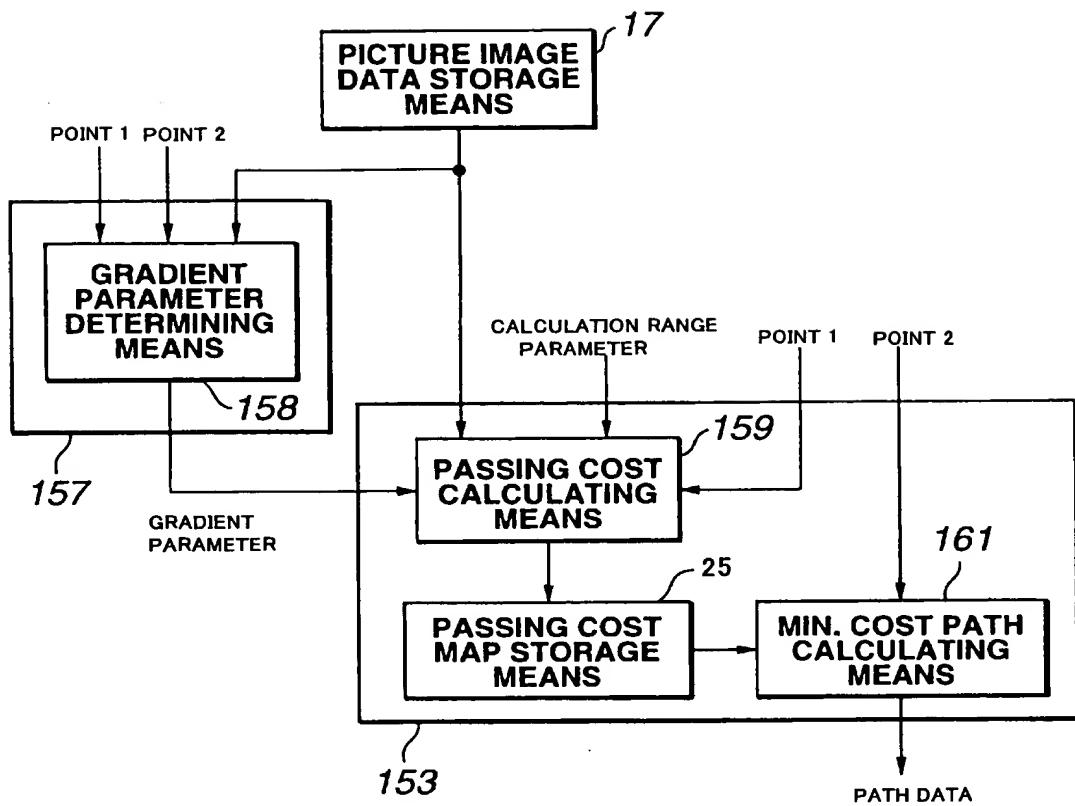
[FIG. 20]



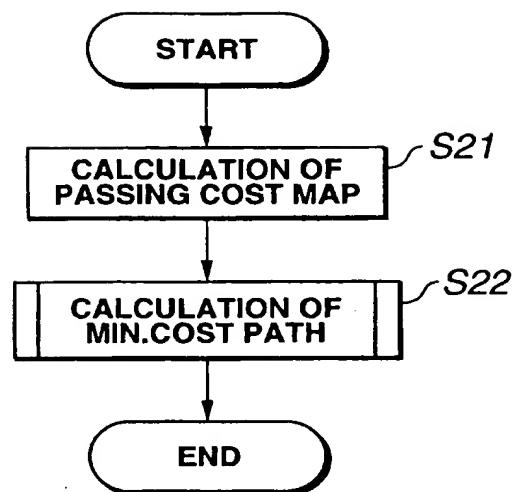
[FIG. 21]



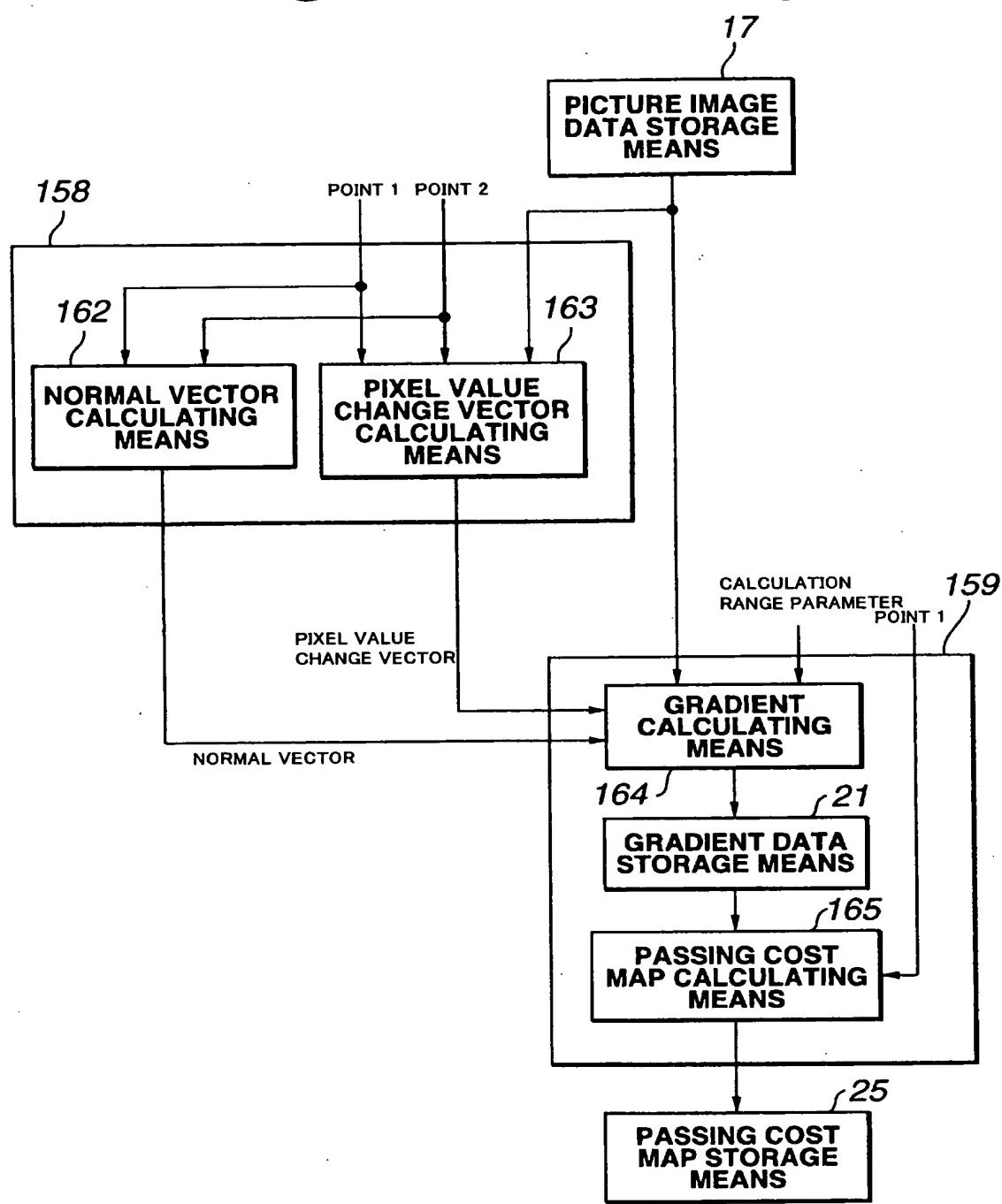
[FIG. 22]



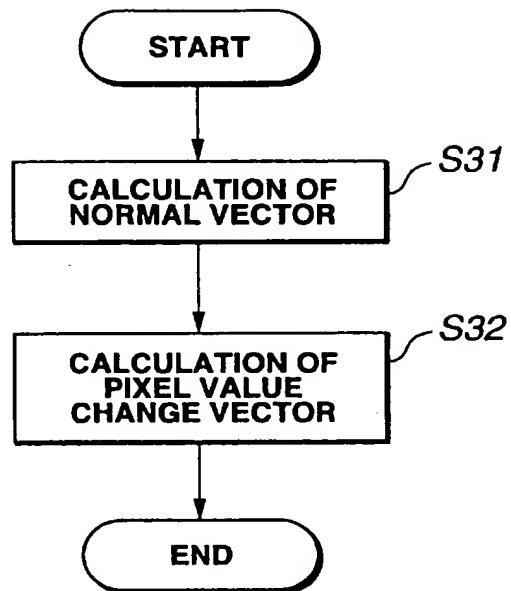
[FIG. 23]



[FIG. 24]

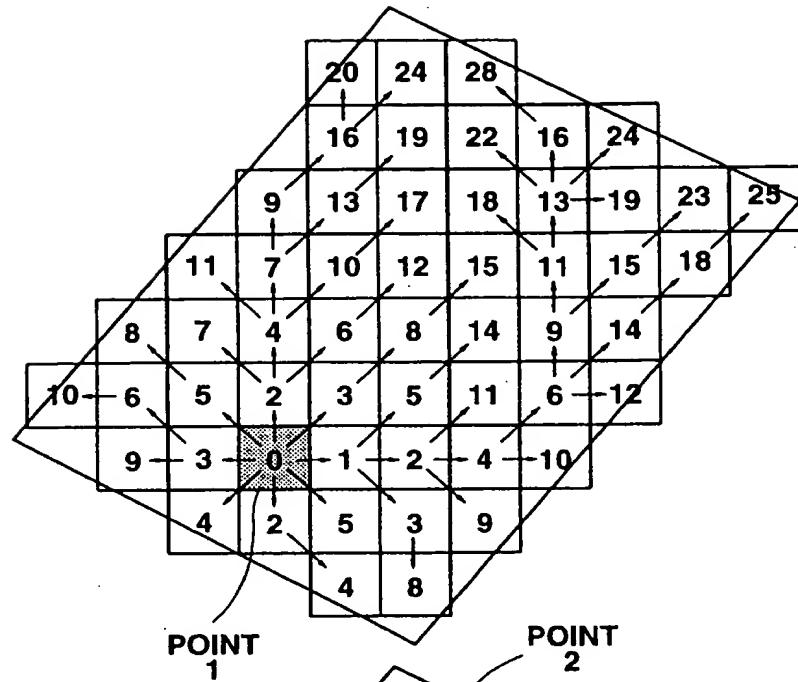


[FIG. 25]

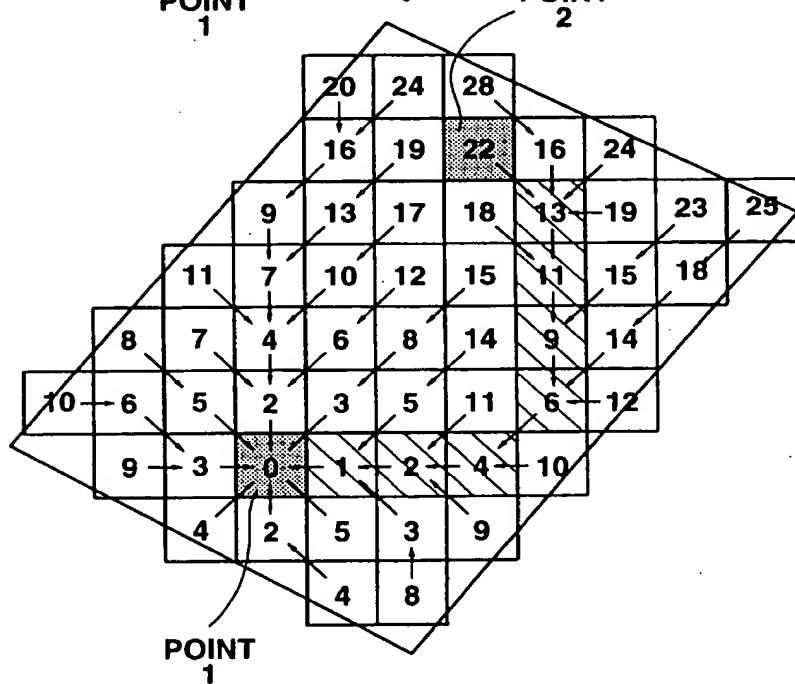


[FIG. 26]

(A)



(B)



The Live-Wire 2-D dynamic programming (DP) graph search algorithm is as follows:

**Algorithm: Live-Wire 2-D DP graph search.**

**Input:**

$s$  {Start(or seed) pixel.}  
 $l(q,r)$  {Local cost function for link between pixels  $q$  and  $r$ .}

**Data Structures:**

$L$  {List of active pixels sorted by total cost (Initially empty).}  
 $N(q)$  {Neighborhood set of  $q$  (contains 8 neighbors of pixel).}  
 $e(q)$  {Boolean function indicating if  $q$  has been expanded/processed.}  
 $g(q)$  {Total cost function from seed point to  $q$ .}

**Output:**

$p$  {Pointers from each pixel indicating the minimum cost path.}

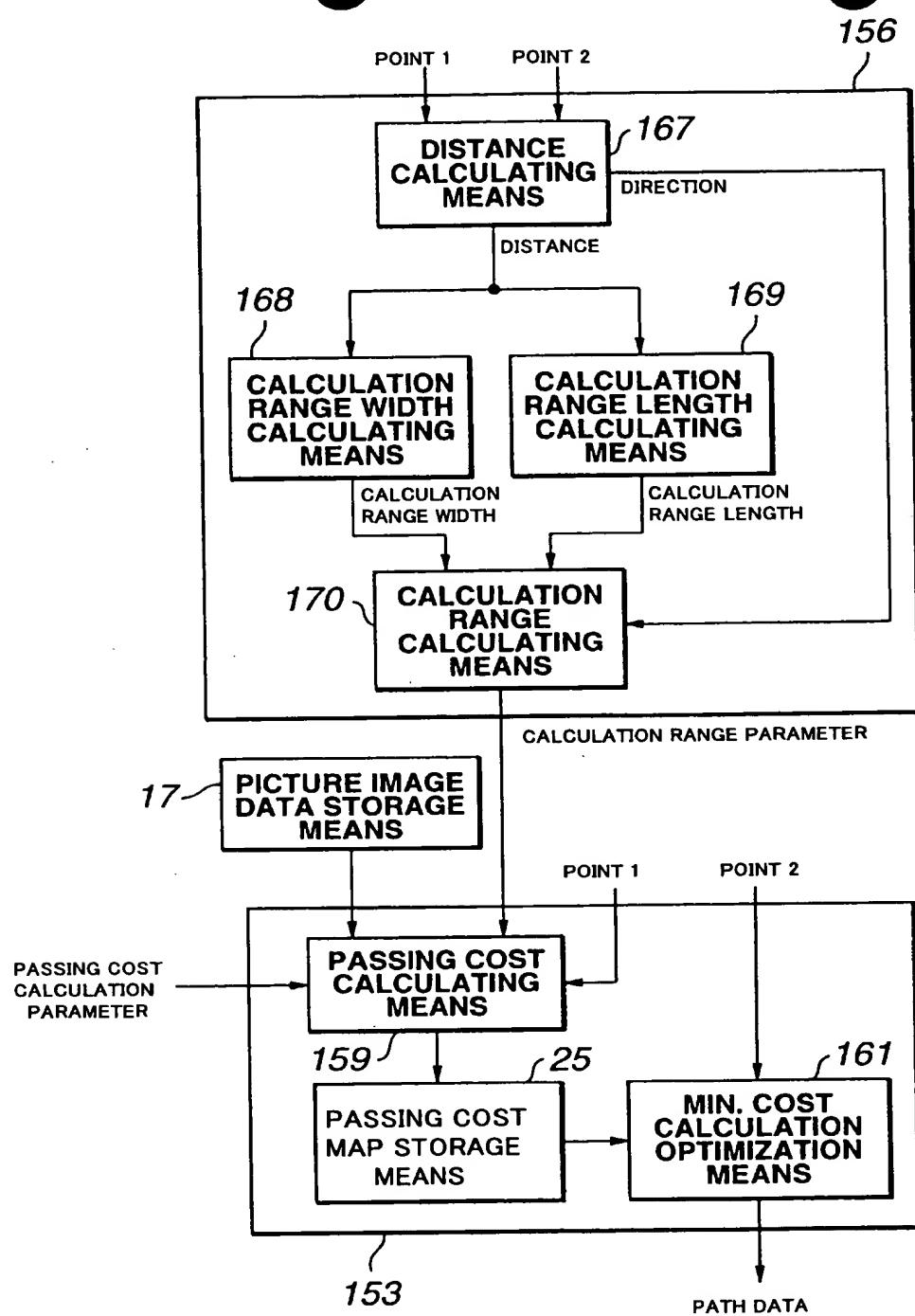
**Algorithm:**

```

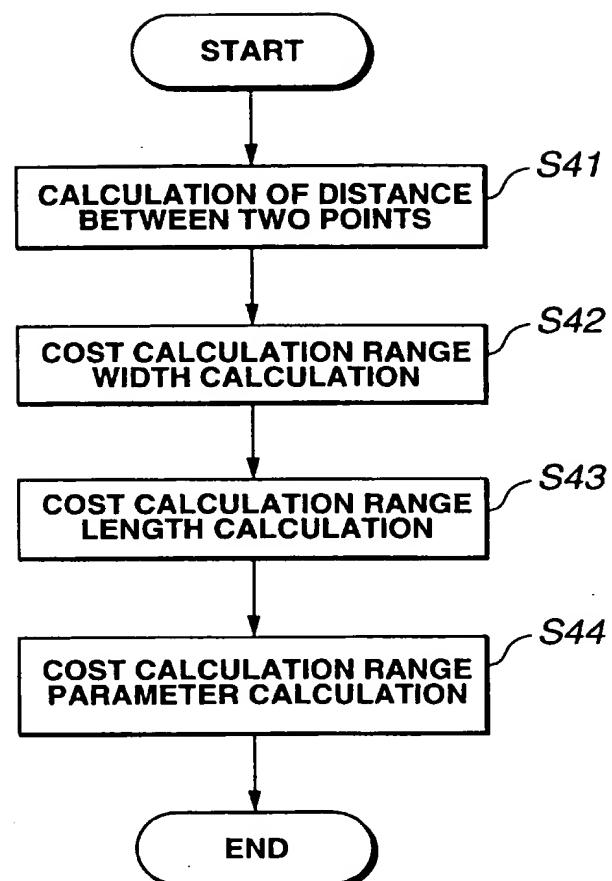
g(s)=0; L=s;                                {Initialize active list with zero cost seed pixel.}
while L!=NULL do begin                      {While still points to expand:}
  q=min(L);                                 {Remove minimum cost pixel  $q$  from active list.}
  e(q)=TRUE;                                {Mark  $q$  as expanded(i.e.,processed).}
  for each  $r \in N(q)$  such that not  $e(r)$  do begin
    gtmp=g(q)+l(q,r);                      {Compute total cost to neighbor.}
    if  $r \in L$  and  $gtmp < g(r)$  then          {Remove higher cost neighbor's}
      r=L;                                    { from list}
      if !( $r \in L$ ) then begin                {If neighbor not on list,}
        g(r)=gtmp;                            { assign neighbor's total cost,}
        p(r)=q;                               { set (or reset) back pointer,}
        L=r;                                  { and place on (or return to) }
        { active list.}
      end
    end
  end
end

```

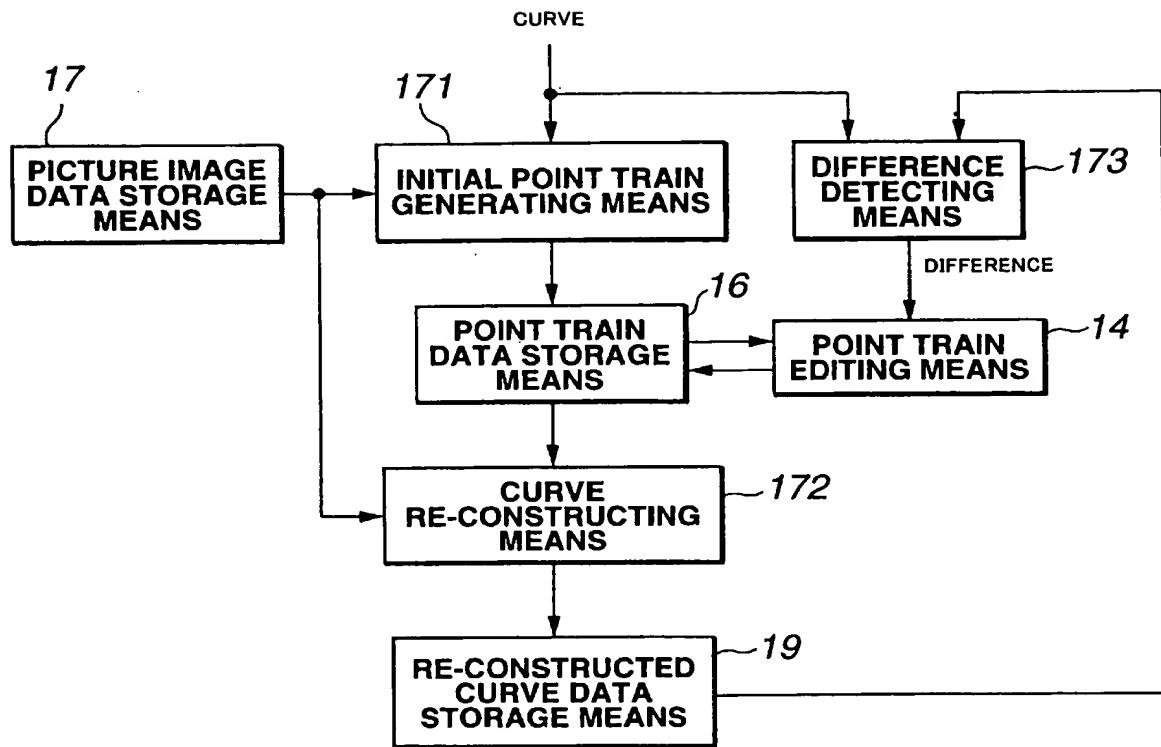
[FIG. 28]



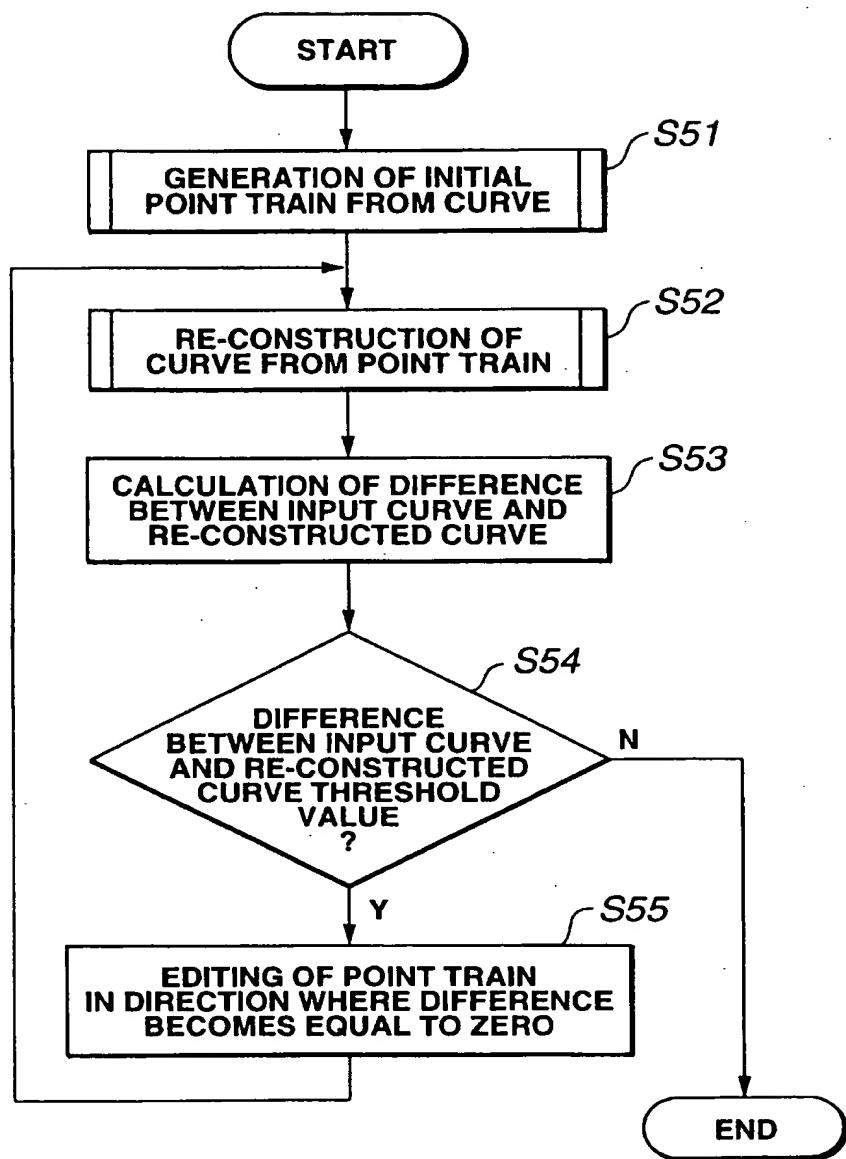
[FIG. 29]



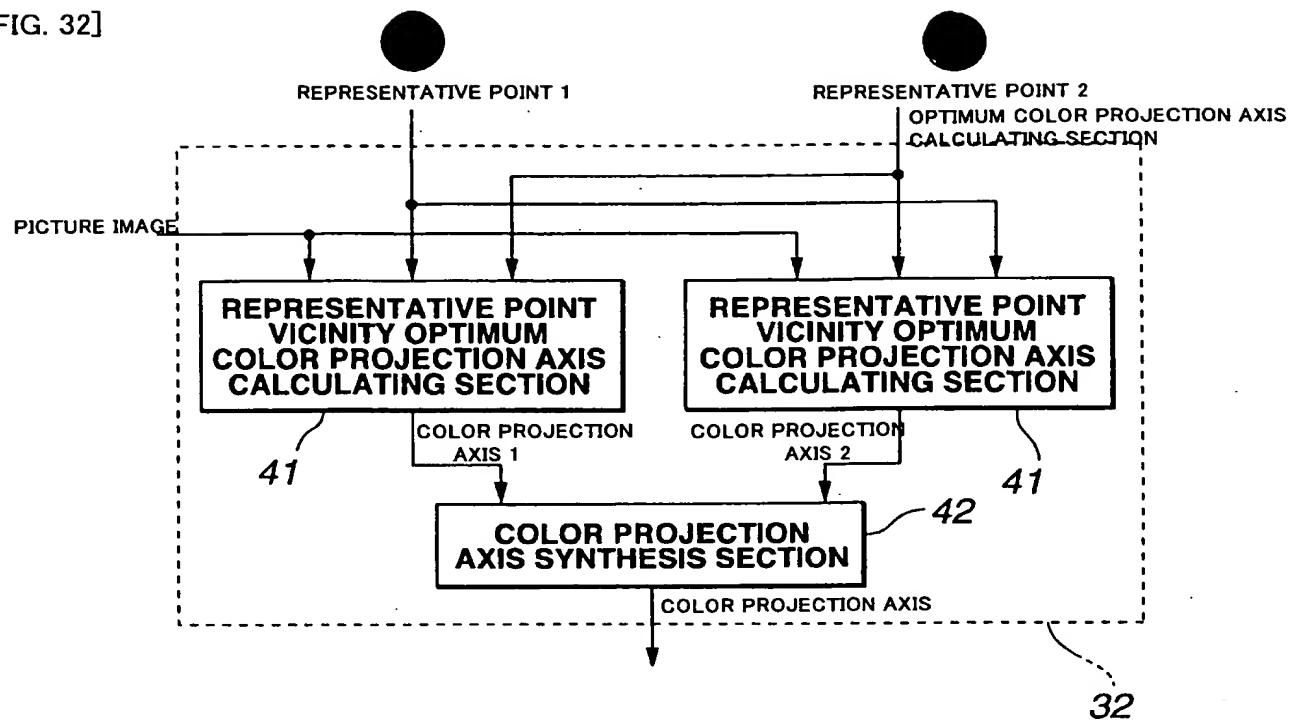
[FIG. 30]



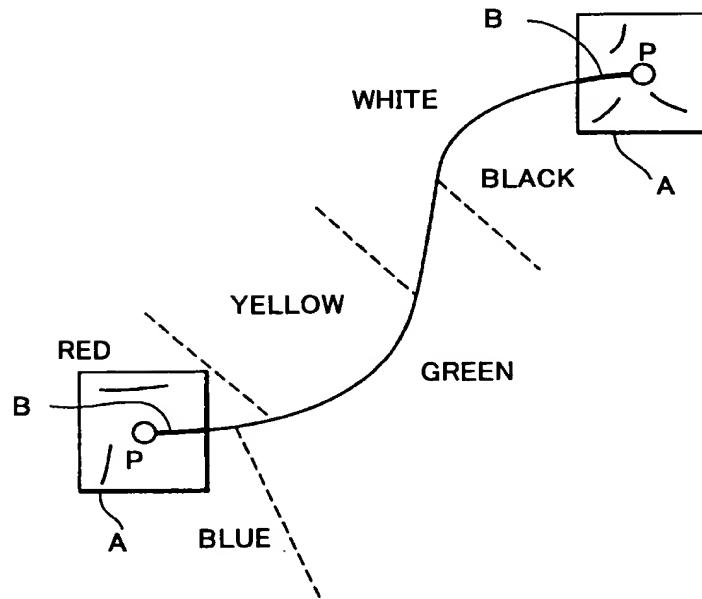
[FIG. 31]



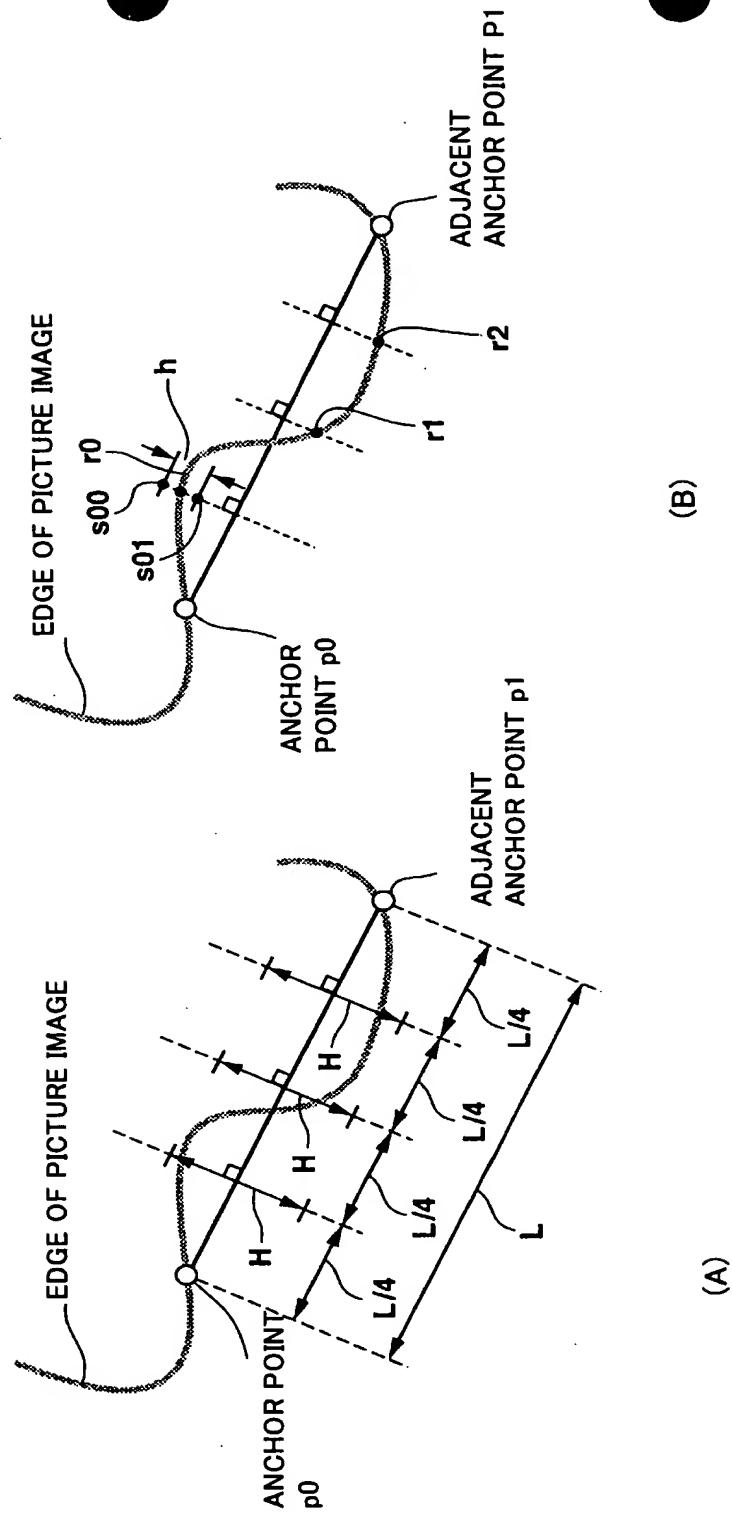
[FIG. 32]



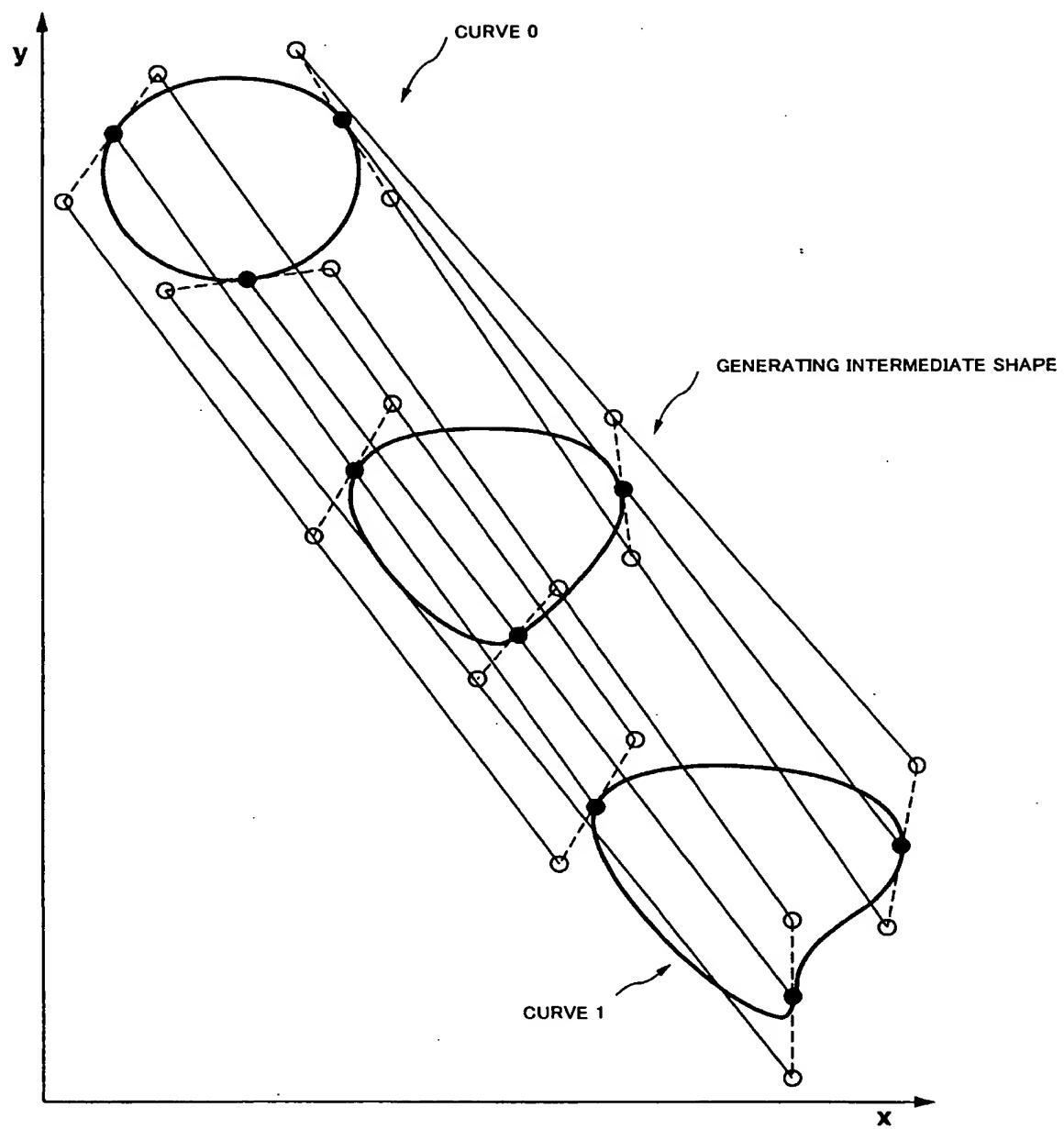
[FIG. 33]



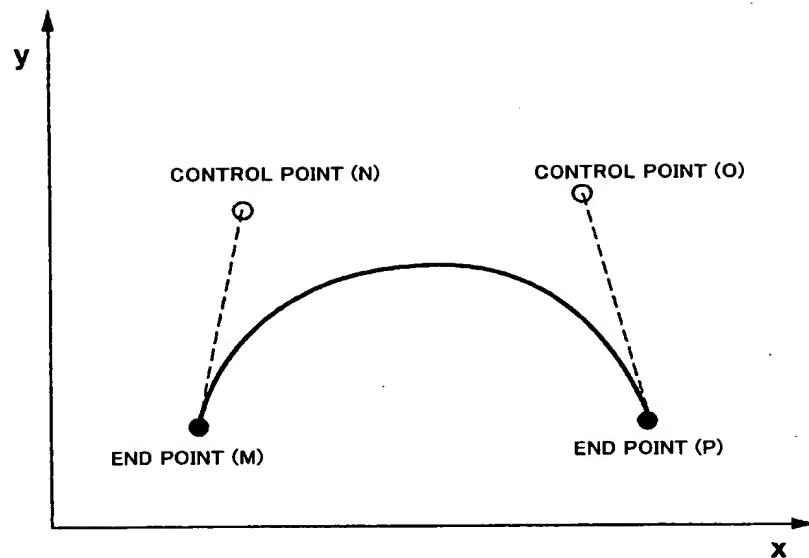
[FIG. 34]



[FIG. 35]



[FIG. 36]



[FIG. 37]

